Desktop haptic virtual assembly using physically-based part modeling

Brad M. Howard
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
https://lib.dr.iastate.edu/rtd/18810

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Desktop haptic virtual assembly using physically-based part modeling

by

Brad M. Howard

A thesis submitted to the graduate faculty in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Mechanical Engineering

Program of Study Committee:
Judy M. Vance (Major Professor)
James H. Oliver
Chris J. Harding

Iowa State University
Ames, Iowa
2005
This is to certify that the master’s thesis of

Brad M. Howard

has met the thesis requirements of Iowa State University
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>vii</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Thesis Organization</td>
<td>3</td>
</tr>
<tr>
<td>CHAPTER 2. VIRTUAL REALITY ASSEMBLY APPLICATIONS</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Full Scale VR Assembly Applications</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Haptic Desktop VR Assembly Applications</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Physically-Based Modeling and Haptics</td>
<td>10</td>
</tr>
<tr>
<td>2.4 Research Focus</td>
<td>12</td>
</tr>
<tr>
<td>CHAPTER 3. PROGRAM STRUCTURE</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Challenges</td>
<td>13</td>
</tr>
<tr>
<td>3.2 System Hardware</td>
<td>14</td>
</tr>
<tr>
<td>3.2.1 Haptic Device/s</td>
<td>14</td>
</tr>
<tr>
<td>3.2.2 Stereo Viewing</td>
<td>15</td>
</tr>
<tr>
<td>3.2.3 PC Systems</td>
<td>16</td>
</tr>
<tr>
<td>3.3 Software Tools and Libraries</td>
<td>17</td>
</tr>
<tr>
<td>3.3.1 Virtual Reality Platform</td>
<td>17</td>
</tr>
<tr>
<td>3.3.2 Haptic Toolkit</td>
<td>18</td>
</tr>
<tr>
<td>3.3.3 Physics Engines</td>
<td>18</td>
</tr>
<tr>
<td>3.3.3.1 Dynamics Engine</td>
<td>19</td>
</tr>
<tr>
<td>3.3.3.2 Collision Engine</td>
<td>20</td>
</tr>
<tr>
<td>3.3.3.3 OPAL Wrapper</td>
<td>22</td>
</tr>
<tr>
<td>3.3.4 Model Format</td>
<td>23</td>
</tr>
<tr>
<td>3.4.1 Configuration and Model Types</td>
<td>24</td>
</tr>
<tr>
<td>3.4.2 Virtual Coupling and Object Manipulation</td>
<td>26</td>
</tr>
<tr>
<td>3.4.2.1 Virtual Spring Equations</td>
<td>27</td>
</tr>
<tr>
<td>3.4.2.2 OPAL Tools for Virtual Coupling</td>
<td>29</td>
</tr>
<tr>
<td>3.4.2.3 Realistic Part Manipulation</td>
<td>30</td>
</tr>
<tr>
<td>3.4.3 Mapping Haptic Device</td>
<td>31</td>
</tr>
<tr>
<td>3.4.4 Application Flowchart</td>
<td>32</td>
</tr>
<tr>
<td>3.4.4.1 Haptics Thread</td>
<td>32</td>
</tr>
<tr>
<td>3.4.4.2 Physics Thread</td>
<td>34</td>
</tr>
<tr>
<td>3.4.4.3 Graphics Thread</td>
<td>35</td>
</tr>
<tr>
<td>CHAPTER 4. SYSTEM RESULTS AND ANALYSIS</td>
<td>36</td>
</tr>
<tr>
<td>4.1 Application Interaction</td>
<td>36</td>
</tr>
<tr>
<td>4.1.1 Model Preparation</td>
<td>36</td>
</tr>
</tbody>
</table>
4.1.2 Manipulating and Selecting Models ...................................................... 37
4.1.3 Interaction Modes ............................................................................. 39
4.2 Program Testing ..................................................................................... 41
  4.2.1 General Assembly ........................................................................... 41
  4.2.2 Collision Detection Testing ............................................................ 43
  4.2.3 Model Loading Tests .................................................................... 45

CHAPTER 5. CONCLUSIONS AND FUTURE WORK ........................................... 47
  5.1 Conclusions ..................................................................................... 47
  5.2 Future Work .................................................................................... 48

BIBLIOGRAPHY ......................................................................................... 51
LIST OF FIGURES

Figure 2.1. (a) MIVAS [18] and (b) CyberGrasp™ hardware (Immersion homepage: www.immersion.com) ................................................................. 5
Figure 2.2. REVIMA and LHlfAM hardware [24]. ................................................ 7
Figure 2.3. VEDA and application hardware [32]. ............................................... 8
Figure 2.4. (a) VShop [37] (b) and Rutgers Master II hardware (The University of New Jersey Rutgers VR lab homepage: www.caip.rutgers.edu/vrlab) ................. 10
Figure 3.1. (a) PHANTOM Omni, (b) PHANTOM Desktop, (c) PHANTOM Premium 1.0, (d) PHANTOM Premium 1.5, (e) PHANTOM Premium 3.0 [33] ........................................... 15
Figure 3.2. Displaying stereo images with quad-buffered page-flipping .............. 16
Figure 3.3. Application software packages/libraries ........................................... 17
Figure 3.4 (a) Original OBJ parts and (b) corresponding NovodeX convex hulls .... 21
Figure 3.5. OPAL integration and simulation time steps ..................................... 22
Figure 3.6. GLM library OBJ loading options .................................................. 24
Figure 3.7. Application infrastructure ............................................................. 26
Figure 3.8. Virtual coupling spring damper system .......................................... 27
Figure 3.9. Unrealistic and realistic physics for a manipulated part ...................... 30
Figure 3.10. Transformation pipeline for mapping haptic device ....................... 31
Figure 4.1. OBJ model conversion paths ....................................................... 37
Figure 4.2. Primitive representation (left) and OBJ tri-mesh representation (right) ... 37
Figure 4.3. The application using stereo vision and dual PHANTOM interaction . 38
Figure 4.4. A virtual wrench (a) before colliding with haptic handle, (b) while colliding with haptic handle, (c) after being selected, (d) and during dual Omni manipulation .... 39
Figure 4.5. Dual haptic interaction with multiple parts .................................... 39
Figure 4.6. Stable tower of dynamic blocks ................................................... 40
Figure 4.7. Assembling primitives to evaluate force feedback ......................... 42
Figure 4.8. Bolt drop collision test ............................................................... 43
LIST OF TABLES

Table 3.1. Computer specifications ................................................................. 17
Table 3.2. Summary of part class characteristics ........................................ 25
Table 4.1. Collision detection test results .................................................. 44
ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Judy Vance, for providing me with an extremely interesting and challenging research project. Her knowledge and advice over the past two years is greatly appreciated.

I would like to thank my other committee members Dr. James Oliver and Dr. Chris Harding for their feedback and involvement with my research. I would also like to thank members of my research group and other VRAC colleagues who have offered advice or technical support.

Last, I would like to thank my family who has always supported and believed in me every step of the way. I could not have done it without you all.
Virtual reality (VR) refers to an immersive, computer-generated three-dimensional (3-D) environment that allows direct manipulation of virtual objects. VR is becoming a popular engineering design tool because of its ability to provide intuitive interaction with computer-generated models and data. As the price of the VR hardware/software becomes more affordable, industry has increased its use of this technology in a number of diverse areas such as medicine [1], education [2], and urban planning [3]. The goal of this research is to investigate the feasibility of using a desktop haptic virtual environment as a design tool for evaluating assembly operations.

Virtual assembly, as it applies to this research, is the ability of the user to intuitively assemble and manipulate computer aided design (CAD) part data within an immersive computer-generated environment. Immersion can be defined as the presentation of sensory cues that convey to users that they are surrounded by a three-dimensional (3-D) computer-generated environment [4]. The immersive aspect of VR offers more intuitive methods to interact with 3-D CAD data than the conventional two-dimensional (2-D) mouse and keyboard input devices. Stereo vision, head/hand position tracking, and visual images displayed on multiple surrounding projection screens are all examples of immersive VR techniques.

Current CAD software packages do provide some tools to evaluate assembly and maintenance operations such as sophisticated animations to help plan part dis/assembly sequences. Tolerance checks can also be performed to analyze if parts will actually fit together. However, these features are not sufficient enough to solve most problems which occur during assembly or when maintenance operators must really interact with a product. Among the problems that CAD packages do not handle are identifying awkward reach angles, insufficient tooling clearance, or the need for additional fixturing.

If virtual assembly is to be successful at identifying these assembly issues before physical prototypes are built, then digital parts need to react in the same way real parts would react. Achieving realistic part motion means that the virtual objects must behave according to the physical laws of the real world by exhibiting properties such as mass, inertia, and
surface friction. Virtual parts need to demonstrate real dynamic properties both while the user manipulates them and when the parts come in contact with each other.

Increasing the amount of immersion or “realism” in a virtual environment is directly related to the amount and fidelity of the sensory cues invoked [5]. While many VR systems have incorporated the human senses of sight and sound, adding the sense of touch or “haptic” feedback is one interaction method gaining more popularity. The word haptic comes from the Greek word *haptesthai* which means to touch [6]. Haptic technology has been combined with graphical displays to help simulate a number of tasks including medical surgery [7], vehicle operation [8], and operator training [9]. Recently, however, force feedback has also been used to help simulate more complicated tasks such as assembly and disassembly operations.

### 1.1 Motivation

Virtual prototyping refers to the use of virtual reality to obtain design evaluations while a product is still in digital form. In a virtual prototype, the VR environment simulates the relevant characteristics of a product's design as realistically as possible in areas relating to design/engineering, manufacturing, product service environment, and maintenance [10].

Many times the computer product data is readily available, but a physical prototype is needed to completely verify a design. The idea is to replace, at least partly, the need for a physical prototype with a virtual prototype. With virtual prototyping, many alternative designs can be explored while data is still in digital form ultimately leading to a better final design. Dealing with vital decisions and making changes before the design is finalized can substantially reduce the product development life cycle. Eliminating a physical prototype will result in substantial cost savings in the overall design process [11].

Manufacturing assembly, maintenance disassembly, and ergonomic operation of a product are all key design considerations that must be examined. Given that the sense of touch is vital to performing these tasks, force feedback is important when interacting with digital product prototypes. Even with collision detection, without force feedback a user’s hand may still go where a part or tool may not really be able to travel. Research has also shown that adding force feedback to virtual assembly environments increases task efficiency.
times [12] [13]. Training for operations that use the sense of touch are also more accurately simulated by implementing haptic feedback. Testing on subjects has shown that operators feel more secure and can relate better to the real world process when trained on a simulator with haptic feedback than those trained on a simulator with no haptic feedback [9].

Full scale VR systems, if totally immersive, typically require a large amount of space and rather expensive hardware. While this does not make them feasible for an engineer to use as a primary design tool, there is still a place for them in the design process. Typically these larger VR systems are more suited for design reviews where a single group of engineers need to collaborate in order to make the best decision. For a typical design engineer who is concerned with assembly and maintenance tasks on a daily basis, bringing some of the immersive qualities of VR to the desktop will greatly enhance his/her ability to evaluate designs.

There are many challenges to creating a haptically enabled desktop virtual assembly application that an engineer can use as an every day design tool. To create such a system, the tools should be both convenient to use and affordable. Among the technical issues to address are: model loading, part collision detection, separate data update or refresh rates, and accurate dynamic modeling of part behavior. This research will investigate a number of haptic assembly applications to determine which tools are the most suitable.

1.2 Thesis Organization

Chapter 1 provides an introduction and the motivation for the research. Chapter 2 discusses a literature review of current virtual reality assembly applications with an emphasis on haptically enabled desktop systems. A brief discussion of physically-based and constraint-based modeling is presented. The challenges of this research project and the structure of the final application including the software libraries and hardware follow in Chapter 3. In Chapter 4, a discussion of the results is presented. Chapter 5 contains research conclusions and suggestions for future work.
CHAPTER 2. VIRTUAL REALITY ASSEMBLY APPLICATIONS

A key component to a realistic VR assembly application is simulating the physical behavior of parts. The two basic methods for simulating physical part behavior are physically-based modeling and constraint-based modeling. In physically-based modeling, Newtonian physics is used to describe the motion of objects in order to model part interaction. These equations are solved each simulation time step based on the forces and torques applied to objects. In constraint-based modeling, certain assembly constraints relating to the geometric properties of objects are identified. For example, part surfaces can be mated together or center line axes aligned. Each constraint reduces the degrees-of-freedom (DOF) of a part in order to simulate real life behavior. Today, most CAD software packages use the constraint-based modeling method to define relationships between parts or created sub-assemblies [14]. VR assembly systems have made use of physically-based modeling, constraint-based modeling, or a combination of the two methods.

This chapter presents a literature review of VR assembly applications. The discussion presents the systems in two categories: full scale VR applications and haptic desktop VR applications. The capabilities of each system as well as the hardware/software tools used were explored to analyze which are most suitable for use in developing an affordable haptic desktop VR application.

2.1 Full Scale VR Assembly Applications

Full scale VR assembly applications are those systems which provide a high sense of immersion but have some limiting factor that makes them unsuitable for operation on a desktop personal computer (PC) system. These limitations range from the space requirements needed for a projection screen system to an expensive or custom made piece of hardware. Although each of these applications has limiting factors, the components of each system may still be useful.

Jayaram et al. [15] have developed one of the more well known full scale VR assembly applications called VADE (Virtual Assembly Design Environment) at Washington State University. VADE displays the virtual assembly environment on either a head mounted display or a single-pipe projection screen system. One or two-handed assembly operations
can be performed using Flock of Birds tracking and CyberGlove®/s. The CyberGlove® can detect bend angles to accurately represent the hand in a virtual environment. Collisions are detected and dynamic interaction of parts, tools, and the virtual environment is achieved. VADE can import parts from CAD packages such as Pro/E and model part behavior based on constraints. Since constraints are used to model part behavior, no reaction forces are calculated and haptic behavior between parts is not available. Recently, a method for collision contact modeling between triangular mesh parts using RAPID™ was implemented for VADE. While this modeling method increased the system’s dynamic simulation capabilities, only simple convex geometry such as a sphere and cylinder were explored [16].

Similar research to VADE has been conducted at Zhejiang University by Wan H. et al. [17] [18] in creating MIVAS (A Multi-Modal Immersive Virtual Assembly System) illustrated by Figure 2.1 (a). Like VADE, MIVAS incorporates tracking devices, a CyberGlove®, and constraints from Pro/Engineer CAD software package to aid in the assembly/disassembly of parts. However, MIVAS provides additional fidelity by allowing the user to feel the size and shape of an object with force feedback from a CyberGrasp™ haptic device shown in Figure 2.1 (b). The CyberGrasp™ is a lightweight, force-reflecting exoskeleton that fits over the CyberGlove® and adds resistive force feedback to each finger. Since the haptic feedback is body-grounded, forces are not simulated when parts collide. Part-to-part collision detection is achieved using RAPID™ while hand-to-object collision utilizes Voxmap-PointShell (VPS). A four wall immersive CAVE™ environment is used to display stereo images of the parts being assembled.

![Figure 2.1. (a) MIVAS [18] and (b) CyberGrasp™ hardware (Immersion homepage: www.immersion.com)](image-url)
Johnson and Vance at Iowa State University developed another full scale VR assembly application called VEGAS (Virtual Environment for General Assembly) [19] and have tested it with complex models from industry. VEGAS can load in models using the geo file format for graphics and VPS software for collision detection. Images can be displayed in a CAVE™ environment and part interaction performed with a position tracked wireless mouse. Kim and Vance [20] expanded the functionality of VEGAS by adding data glove interaction and implementing physically-based modeling for parts using VPS. This research included an in-depth study of several collision detection and physically-based modeling libraries for part assembly [21]. Although VEGAS can handle collision detection and accurate part behavior for complex geometry, no haptic interaction has been implemented.

Kim and Vance [22] also developed NHE (Network Haptic Environment) to enable assembly tasks to be evaluated by individuals in remote locations. The application is similar to VEGAS in that VPS is used for physics and collision detection but haptic part interaction is also implemented. Force feedback is achieved by using several SensAble PHANTOM haptic devices and the GHOST® SDK software library. A combination of server-client and peer-to-peer system architecture along with socket communication is used to consistently keep track of data. However, the computational capabilities of each client computer often caused inconsistency in shared data leading to unrealistic force feedback [22].

Other research conducted by Fischer and Vance [23] at Iowa State University resulted in a method to map the PHANTOM haptic device to an immersive CAVE™ environment. Several software packages including VR Juggler, GHOST®, and VPS were combined to explore the benefits of haptic feedback in performing various tasks. Two example scenarios of exploring a NURBS (Non-Uniform Rational B-splines) surface and installing an aircraft rudder pedal assembly were achieved [23].

CIET, a Spanish non-profit research organization, has created a virtual hardware/software system called REVIMA (Virtual Reality for Maintainability) [24] [11], shown in Figure 2.2, which can be used to investigate maintenance and assembly operations for aircraft engines. The group developed a hardware device called LHIfAM (Large Haptic Interface for Aeronautic Maintainability) which can measure 6 DOF input and provides 3 DOF force feedback while allowing access to the entire length of a virtual aircraft
engine. CAD models can be loaded into the application for graphics while a voxel-based method is used for collision detection. Digital models of aircraft engines can be viewed on a single projection screen (without stereo) and be haptically manipulated.

![Figure 2.2. REVIMA and LHIfAM hardware [24]](image)

### 2.2 Haptic Desktop VR Assembly Applications

There have been numerous applications created to simulate VR assembly applications using a desktop PC. These applications range from virtual factory layout simulations [25] to evaluating manufacturing assembly sequences with augmented reality [26]. Here the focus is limited to those desktop VR assembly systems which are haptically enabled.

Bert Bras et al. [27] [28] [29] [30] [31] from the Georgia Institute of Technology have developed a well known haptically enabled desktop simulation environment called HIDRA (Haptic Integrated Dis/Re-assembly Analysis). Haptic force feedback is implemented using a dual PHANTOM setup and GHOST® software library. The user can interact with parts by using the two PHANTOMs to “pinch” and pick up objects. Simplified rigid body dynamics, constraint maintenance, and velocity slow down compensation all help speed up the simulation calculations but distract from the application’s realism. CAD models can be imported via VRML file format. The V-Clip or SWIFT++ collision detection libraries use Qhull to recreate the models with convex hulls for collision detection. A quantitative test of the time and difficulty to assemble a bolt in a hole using both V-Clip and SWIFT collision detection libraries was tested with results showing that V-Clip is better suited for the application [30]. Testing has also been done on HIDRA to compare weight sensation in VR with real life scenarios. Results showed that the tolerance for recognizing weight differences
in VR was undetermined and that weight differences were not recognized as fast or as effectively when compared to the real world [31].

Similar research has been conducted by Gupta et al. [32] at the Massachusetts Institute of Technology in creating a multi-modal desktop system called the VEDA (Virtual Environment for Design Assembly) shown in Figure 2.3. The system uses the same dual PHANTOM setup as Bras’s work with HIDRA but limits the virtual environment and haptic feedback to two dimensions in order to simplify dynamic calculations. VEDA uses force feedback, physically-based modeling, accurate collision detection, sound cues, and stereo vision to provide a more realistic virtual assembly experience. In a study, subjects were to complete the identical assembly task of placing a bolt into a hole using both an actual assembly process and VEDA. Results showed VEDA assembly times to be roughly twice as long as actual assembly times [32].

![Figure 2.3. VEDA and application hardware [32]](image)

The previous desktop examples have all used haptic devices with 3 DOF force feedback or devices that lack torque feedback. SensAble Technologies Inc. does commercially manufacture haptic devices with both force and torque feedback such as the PHANTOM Premium 1.5/6DOF and 3.0/6DOF [33]. Boeing’s Mathematics and Computing Technologies Division (M&CT) has also teamed up with SensAble to produce a desktop virtual prototyping application that tests the PHANTOM 3.0/6DOF with Boeing’s VPS collision and contact response software [34]. When testing the application, a teapot model was manipulated in a complex scene of several thousand polygons while the user received
both force and torque feedback. The torque feedback added to the realism of the simulation as well as increased the ability to orient objects in tight spaces.

Haptic technology has also been applied to CAD packages to provide the ability to feel and manipulate CAD data from a desktop PC. The European research group LABEIN has developed a haptic CAD environment by integrating dual PHANTOM interaction with the DATum CAD software [35]. Within the application a user can touch, move, and detect collisions between parts in order to simulate assembly and maintenance tasks. Manipulated parts are restricted from penetrating each other while audio sounds alert the user of collision events. No real time physics or dynamics are applied in the application.

SensAble has developed a product called Freeform® which can import/export many industry standard model file formats allowing the user to haptically manipulate models with PHANTOM haptic devices [36]. Freeform® models are made out of virtual “clay” which users can “feel” and manipulate using digital sculpting tools. No interaction between modeled parts is implemented in the Freeform® software however.

Pere et al. [37] at Rutgers University developed a haptic device called the Rutgers Master II (RMII) (Figure 2.4 (b)) for performing assembly tasks in a PC based virtual assembly workshop called VShop (Figure 2.4 (a)). The RMII consists of a position tracked glove with four pneumatic actuators that can apply force to the user’s finger tips as well as track the user’s hand gestures. Objects in the virtual workshop can be touched or grasped while force feedback is provided by the RMII. VShop displays graphics on a PC using OpenGL and creates the virtual environment using the World-ToolKit software. Navigation through the environment is done via mouse interaction. Gravity and collision detection are also implemented but with minimal fidelity.
From the literature review, it is evident that each system seems to lack characteristics in one area or another for the ideal haptically enabled desktop VR application. While the full scale systems usually have some custom made or expensive piece of equipment, the haptic desktop systems tend to fall short in realistic part representation. A few desktop systems did not implement any part dynamics and those that did tended to use over-simplified models or models that only apply to specific circumstances. These results indicate there is a need for a system that utilizes affordable equipment while still being able to provide realistic part interaction. The goal of this research is to investigate the feasibility of using such a desktop haptic virtual environment for evaluating assembly operations.

2.3 Physically-Based Modeling and Haptics

Physically-based and constraint-based modeling methods each have distinct advantages and disadvantages. Physically-based modeling methods can handle a greater number of situations without pre-defining constraints for each particular circumstance. However, physically based-modeling algorithms require more computation time to accurately detect collisions and to solve for all the forces acting on an object. One severe drawback to constraint-based modeling is that each part must undergo a preprocessing step where the constraints are identified [15]. Information about the constraint data must be extracted from CAD assembly models and supplied to the simulated parts. Manually applying these
constraints can involve an extremely large amount of model preparation work prior to simulation. Extracting and applying constraint data from CAD packages has been implemented in related research [15] [17] [18].

Physically-based modeling methods can be divided into three categories: the penalty force method, the impulse method, and the analytical method. In the penalty force method, a penalty force pushes apart two colliding bodies. This is most commonly accomplished with a virtual spring and damper system attached to the point of contact between two colliding objects. As the objects move toward each other the spring attempts to push them apart [14]. The spring force is calculated using Hooke’s Law which results in a restoring force that is directly proportional to the penetration depth of the two objects. The penalty method requires a computation time on the order of $O(n)$ [38] where $n$ is the number of contacts. Since this method does not require a large amount of computation time and is relatively easy to implement, it is quite commonly used with haptics.

Xiao and You [39] at the University of North Carolina have developed a physics model using the penalty method that can be used to haptically manipulate two convex polyhedra. The method uses information prior to a collision event to accurately simulate responses taking into account friction and gravity. A more accurate penalty method has been combined with haptics by Hasegawa and Sato [38] at the Tokyo Institute of Technology. This method produced very realistic results for surface friction since it takes into account the surface contact area in addition to just contact points which are implemented by most penalty methods.

The impulse method uses Newton’s second law of momentum conservation to calculate forces on objects when a collision takes place. The calculated impulse forces are then applied to the objects to keep them from interpenetrating. This method works rather well for objects that are temporarily colliding but less well for continuous contacts such as resting or sliding. Beeling and Colgate [40] at Northwestern University have created an impulse-based simulation as a general purpose multi-body simulator with haptic force feedback. Since the impulse method does not readily handle continuous contacts, the method was not able to realistically simulate frictional forces.
The analytical method [41] [42] solves for forces based on conditions of non-penetration constraints and equations of motion. This method uses ideally rigid objects in quasi-steady equilibrium and applies the concept of an impulse for non-equilibrium motion. Linear programming is used to solve for exact surface boundary constraints. The computation time for this method is relatively long and is on the order of \( O_n^3 \) where \( n \) is the total number of contacts in a system [38]. This longer computation time creates problems for a large number of contacts when trying to maintain haptic rates. Because of this, the analytical method has not been implemented with haptic force feedback.

### 2.4 Research Focus

As previously stated, the goal of this research is to investigate the use of a haptic desktop VR system to perform assembly analysis. It is important that the VR assembly application utilize tools that are affordable by industry. This research will concentrate on exploring the benefits and limitations of combining the physically-based modeling of parts with haptic force feedback. Prior research has shown a trend to write separate in-house physics simulations for haptically enabled applications. Often times the physics for such applications are over simplified and not robust. For VR assembly applications, unrealistic physics can result in erroneous part behavior, which ultimately hinders the usefulness of the simulation.

Today there are many commercial and open source physics engines that have been developed by experts in the field. Physics engines are software libraries that provide the physically-based simulation of objects. Developing a robust physics engine is no simple task and requires expertise in a diverse set of broad and complex subjects [43]. In some sense, writing a physics engine from scratch seems futile when there are many commercial and open source alternatives. This research will make use of a suitable physics engine to provide the physically-based modeling of objects for haptic interaction.

The purpose of this research is not to develop any particular piece of software/hardware, but rather to investigate and combine suitable tools to create a functional application. Analysis of the final application and tools used will be presented to provide knowledge for future work in the field of haptic virtual assembly.
CHAPTER 3. PROGRAM STRUCTURE

The application presented in this research combines force feedback, physically-based part modeling, and desktop VR characteristics such as stereo vision. To combine these tools many technical issues were addressed. This chapter presents the rational behind the selection of various components of the VR system and solutions to problems encountered in the system’s development.

3.1 Challenges

Many haptic devices have been developed in recent years with only a few becoming commercially available. Most of these touch enable devices have been developed with specific tasks in mind and are not extendable to other needs. Of the commercial haptic devices, many are still too expensive to use as an everyday design tool and are more suited for strictly research development. The ideal haptic device for this research needs to be affordable, provide quality force feedback, and be relatively easy to program.

To provide realistic force feedback for stiff contacts, forces must be sent to the haptic device at a rate of 1000 times per second (1 kilohertz) or faster. This is because the human body’s kinesthetic sensors can detect changes in motion slower than 1 kHz [44], therefore; a haptic simulation looses fidelity if the force signals fall below this update rate. However, every aspect in a haptic simulation does not need to run this fast. The human eye can only detect image changes at a rate of 30 times per second so graphic frame rates are typically set at 30-60 Hz. Realistic physics simulations require significant computation which inhibits achieving 1 kHz refresh rates. Physics loops are usually set to run around graphics rates or rarely faster (~40-100 Hz). Therefore, displaying graphics and simulating physics at haptic refresh rates can waste valuable computation time. One solution is to provide a separate dedicated thread for each of the physics, graphics, and haptics loops. However, data sharing requires special attention since each thread must access the same constantly changing state variables.

There are also many issues to resolve with physically-based modeling and the accurate representation of part behavior. One problem closely associated with physically-based modeling is that of collision detection. While collision detection is relatively easy for
predefined convex shapes, the difficulty occurs when arbitrary non-convex geometry needs to be simulated. Since not all parts used in an assembly application are necessarily convex, this issue needs to be addressed. Finding adequate software to perform accurate collision detection and simulate rigid body dynamics at fast enough speeds for haptic refresh rates will have to be achieved. File formats and conversion paths must also be identified for loading CAD model data into both physical and graphical part representations.

3.2 System Hardware

Ideally, the selected hardware should be easily integrated into the engineer’s everyday workstation. This means factors such as cost, size, availability, and adaptability of the equipment must all be considered. The equipment used to display stereo graphics includes a Stereographics Emitter, Crystal Eyes shutter glasses, and a high refresh rate CRT Monitor. One or two handed force feedback is obtained by various PHANTOM haptic devices from SensAble Technologies.

3.2.1 Haptic Device/s

In selecting a haptic device, both the cost and quality of force feedback were considered. System hardware such as MIVAS’s CyberGrasp™ [17] [18] and VShop’s Rutgers Master II [37] provide natural forces for gripping objects, but lack force feedback related to part collisions because they are body-grounded devices. These devices do not provide the force feedback required to impede the user’s motion when manipulated objects collide with other objects in the virtual environment. A floor or table-grounded device is required to simulate gravity and provide object-to-object force feedback. Various research groups have produced prototypes of desk-grounded haptic devices [11], with the most common commercial products being the PHANTOM haptic devices from SensAble Technologies Inc. [33].

SensAble is the leading provider of 3D touch-enabled digital solutions for commercial software and academic research [33] [45]. The company offers a variety of PHANTOM devices, shown in Figure 3.1, ranging from the more affordable desktop solutions of the PHANTOM Omni (Figure 3.1 (a)) and PHANTOM Desktop (Figure 3.1 (b)),...
to the higher precision and larger workspace PHANTOM Premium models (Figure 3.1 (c),(d),(e)).

![Figure 3.1](image-url)

**Figure 3.1.** (a) PHANTOM Omni, (b) PHANTOM Desktop, (c) PHANTOM Premium 1.0, (d) PHANTOM Premium 1.5, (e) PHANTOM Premium 3.0 [33]

The PHANTOM Omni was selected as the primary haptic device for this application although all of SensAble’s haptic devices are easily integrated. Many characteristics of the Omni make it ideal for a desktop assembly application including its relatively low cost at approximately $2400. The device limits the input workspace to approximately the range of hand motion pivoting at the wrist. The Omni is easily connected to any computer via IEEE 1394 FireWire® port while many other PHANTOM devices require a parallel port or a data acquisition card. Dual Omni’s can also be attached to a PC by using two fire wire ports or a single port by connecting the haptic devices in parallel. The Omni can input 6 DOF (x, y, z, roll, pitch, yaw) data to a simulation while providing 3 DOF (x, y, z) force feedback with up to 0.75 lbf. (3.3 N). Calibration is done with a stylus-docking inkwell that automatically calibrates the device each time a haptically enabled application is used.

### 3.2.2 Stereo Viewing

Visualizing and interacting with complex 3-D CAD data on a 2-D monitor can be a difficult task. Stereo or 3-D imaging provides the user with a more intuitive interface by conveying depth perception and spatial cues. A method known as quad-buffered page-flipping was used to enable the active stereo vision for this application (Figure 3.2). In this method, separate right and left eye images are alternately displayed on a high refresh rate CRT monitor. Crystal Eyes shutter glasses are worn by the user to block the right eye when the left eye image is display and vice versa. An emitter from Stereographics Corporation synchronizes the eye that is blocked with appropriate image displayed.
Figure 3.2 illustrates how the perceived location of the stereo image is determined. Displaying the same image slightly offset for each eye coupled with the fact that a human’s eyes have slightly different viewpoints allows the displayed images to have depth perception. Images are displayed faster than the human eye can perceive at or above 30 Hz. The two images are blended together as single image whose location appears where the two views cross. Quad-buffered page-flipping requires a monitor with a vertical scan frequency rate of 120 Hz or higher so that images appear fluent and bleeding off alternate images does not occur. VR Juggler, the open source framework for the application, handles the image offset preprocessing.

![Diagram of stereo glasses and image offset]

**Figure 3.2. Displaying stereo images with quad-buffered page-flipping**

### 3.2.3 PC Systems

Application testing was performed on two different computer systems to analyze performance. These systems approximate the higher and lower ends of typical industry workstations for processor speed, memory, and graphics cards. A Dell™ Inspiron 8600 laptop was used to represent the lower end machine while a Dell™ Precision 670 represented the higher end. The laptop met all the fundamental hardware requirements for the application with the exception of a high refresh rate monitor. Table 3.1 provides a summary of the hardware for each computer.
Table 3.1. Computer specifications

<table>
<thead>
<tr>
<th>Computer Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
</tr>
<tr>
<td>Processor</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Graphics Card</td>
</tr>
</tbody>
</table>

3.3 Software Tools and Libraries

Many different software packages/libraries were combined to create the application. Figure 3.3 gives the functionality and hierarchy of the libraries used. When selecting the software tools, factors such as cost, ease of programming, and robustness were all taken into consideration.

![Figure 3.3. Application software packages/libraries](image)

C++ was used as the programming language and Microsoft Visual Studio .NET 2003 as the development environment. SensAble’s OpenHaptics™ SDK toolkit was used to drive the PHANTOM haptic device/s, VR Juggler to provide the virtual environment/graphics framework, OpenGL/GLUT/GLM to render the graphics, ODE/OPAL to simulate part physics, and OPCODE/ODE primitives to perform collision detection. Object oriented programming concepts were used to combine the functionality of each library.

3.3.1 Virtual Reality Platform

The application framework for this research was constructed using VR Juggler, an open source library for creating virtual environments. The VR Juggler Application
Programming Interface (API) was selected over other APIs for many reasons. The software hides many of the lower-level programming details required to develop, test, and debug VR applications [46]. Applications created with VR Juggler are also extendable in that they are independent of device, computer platform, and VR system. Since VR Juggler is open source, the entire software package including source code is available for use completely free of charge. Among the useful tools provided by VR Juggler for this research are the Juggler Configuration and Control Library (JCCL), Gadgeteer device manger, and VR Juggler Portable Runtime Layer (VPR). VR Juggler was created by researchers at Iowa State University and is being actively developed.

3.2.2 Haptic Toolkit

SensAble offers two programming API toolkits to add force feedback to the PHANTOM haptic devices: OpenHaptics™ toolkit and GHOST® SDK. While the GHOST® library has some predefined primitive shapes and simplified rigid body dynamics functionality, the OpenHaptics™ toolkit is structured to integrate third party physics engines. The newest haptic device from SensAble is the low cost Omni. Because the Omni cannot be controlled by GHOST®, OpenHaptics™ toolkit was selected as the haptic programming API for this research.

OpenHaptics™ represents the newest generation of API and device drivers from SensAble. The word “open” implies that the software is similar in programming structure to OpenGL. The toolkit allows lower-level programming access to the PHANTOM devices through the Haptics Device API (HDAPI) or higher-level programming access using the Haptics Library API (HLAPI) which is built on top of the HDAPI functionality. The HLAPI is most useful for adding the sense of touch to existing OpenGL code. Only the HDAPI was utilized for this research since it allows direct access to the scheduler, which manages the high frequency, high priority haptic thread.

3.3.3 Physics Engines

Open Dynamic Engine (ODE) [47] was selected as the physics engine to provide the physically-based modeling of parts for this research. ODE is a stable open source physic engine widely used in the computer gaming community. The engine uses an advanced form
of the penalty method by creating hard contacts or non-penetrating constraints whenever two bodies collide. A highly stable integrator is used to solve for forces as opposed to an exact solution. The focus of ODE is on speed and stability over physical accuracy which makes it ideal for integrating haptics but not adequate enough for quantitative engineering analysis.

Many commercial and open source physics engines are available today. Commercial physics engines such as NovodeX [48] and Havok [49] offer more programming options but the basic functionality required for an assembly application is just as well provided by ODE. The two basic components to any physics engine are the dynamics simulation engine and the collision detection engine.

3.3.3.1 Dynamics Engine

The dynamics solver engine is responsible for solving equations related to the rigid body dynamics of an object. When a body is said to be “rigid” it is assumed that the body is non-deformable and that the shape of an object does not change. Physical properties of an object that are constant with time such as mass, center of gravity position, and inertia are set by the programmer. Based on the forces applied to the rigid body, an integrator solves for the state variables or variables that can change with time such as position, orientation, and linear/angular velocities. Given the values of these state variables, an object’s motion can be simulated. ODE was selected over other physics engines because its numerical integrator is just as stable as most commercial packages and it is available as open source software.

ODE simulates articulated rigid body dynamics by applying constraints, commonly called joints, to the rigid bodies. A variety of different joints can be used to restrict a body’s motion including a temporary contact joint created when two bodies collide. A contact joint is added and removed each time step that a collision event occurs to keep bodies from penetrating each other. All constraints within ODE are governed by the following three joint equations:

\[
J_1 v_1 + \Omega_1 w_1 + J_2 v_2 + \Omega_2 w_2 = c + C\lambda \quad \text{Eqn. (3.1)}
\]
\[
\lambda \geq 1 \quad \text{Eqn. (3.2)}
\]
\[
\lambda \leq 1 \quad \text{Eqn. (3.3)}
\]

The above equations represent vectors of size \( m \times 1 \) where \( m \) is the number of constraints placed on a particular object. The coefficients \( J \) and \( \Omega \) represent \( m \times 3 \).
Jacobian matrices. The variables $v_1, v_2, w_1, w_2$ represent the linear and angular velocity state variables for the first and second colliding objects respectfully. The $\lambda$ variable is an $m \times 1$ constraint force automatically calculated by ODE to ensure the constraint equation holds true. The variable $c$ is a right hand vector controlled by the "error reduction" parameter (ERP) while $C$ is an $m \times m$ matrix called the "constraint force mixing" (CFM) matrix. The values of ERP and CFM can be changed by the programmer to make a particular constraint behave in a desired manner. Since the focus of this research is on physically-based modeling, only contact joints are utilized.

### 3.3.3.2 Collision Engine

The collision detection engine is responsible for providing the dynamics engine contact information when two rigid bodies collide. To use the collision engine in a rigid body simulation, shapes are associated with each rigid body object. The shape data is different from the dynamic information in that it contains an object’s geometrical properties (size, shape, position/orientation, and surface friction) but no dynamical properties (mass, velocity, and acceleration). Each integration time step, ODE’s collision engine calculates the contact points of the colliding bodies and passes them to the dynamics engine. The contact point information is then used to create constraint equations Eqn. (3.1), (3.2), (3.3) to be solved during the simulation. Together, the shape and rigid body information represent all the properties of a simulated object.

ODE’s collision detection engine can represent an object’s shape with primitives (sphere, box, and capsule) or arbitrary shapes using a triangle-mesh data format. The internal collision detection engine is an optional feature of ODE and other collision engine libraries can be implemented so long as they return the correct parameters. Other physics engines, such as NovodeX, have the ability to create convex hulls for collision detection. While convex hulls may be adequate for simulations that do not require high accuracy, realistic part assembly requires a better model. Figure 3.4 shows two relatively simple non-convex CAD models (Figure 3.4 (a)) and the corresponding convex hulls (Figure 3.4 (b)) created using NovodeX. The parts illustrate how important features for assembly are lost such as the
block's hole and the sharply defined edges of the bolts under side when convex hulls are used as the basis for collision detection.

![Figure 3.4 (a) Original OBJ parts and (b) corresponding NovodeX convex hulls](image)

One common method to store CAD model data is as a set of triangles or polygonal surfaces. The individual triangles connect to form a “tri-mesh” when pieced together represent an entire CAD model. To model accurate collision responses, the CAD tri-mesh data should be directly represented. Many current physics engines provide the ability to read in and represent mesh data for simulating collisions and physical responses. However, modeling accurate mesh-to-mesh collision responses is currently still an active research problem being investigated by the physics simulation community.

ODE provides mesh collision detection using a software package called Optimized Collision Detection library (OPCODE) [50]. Although ODE's mesh-to-mesh functionality is fairly new and experimental, it appears to perform just as well as any other physics engine package available today. OPCODE uses a memory-optimized bounding-volume hierarchy of axis-aligned bounding boxes to detect when and where two meshes collide. When compared with similar packages such as RAPID™ [51] and SOLID [52], it is noted that OPCODE uses considerably less memory. Although OPCODE appears to be one of the best mesh collision detection libraries to date, ODE does allow the ability to implement different mesh collision libraries.
3.3.3.3 OPAL Wrapper

For this research, an open source physics engine wrapper called Open Physics Abstraction Layer (OPAL) [53] was used to implement ODE. OPAL offers a higher-level API for lower-level physics engines. The goal is to hide many of the lower-level programming details and adjustable parameters so the user can focus on creating the actual application. The OPAL programmer is supplied with a suite of useful tools built on top of ODE functionality. These tools include intuitive structures such as sensors, motors, and solids. OPAL also provides the ability to load complex objects comprised of these data structures from an XML file format.

Within OPAL, a structure called a simulator encapsulates all the functionality of ODE’s collision detection and dynamic simulation engines. The simulator is responsible for creating, maintaining, and destroying all the simulated objects. Certain parameters relative to the performance of the physics simulation can be altered using the simulator. For example, the integration step size can be set to a desired value. A step size of 0.01 means a simulation will update at a rate of 100 Hz. Figure 3.5 illustrates that a smaller integration step size results in a more accurate simulation or one that is closer to the ideal simulation. However, as the step size decreases, the required computation time increases. The OPAL programmer is responsible for passing the simulator the amount of time $dt$ to simulate ahead in order to provide physics for the next frame. The easiest way to accomplish this is to pass the elapsed time from the previous frame to the simulator each consecutive frame. Figure 3.5 shows two consecutive frame time steps $dt_1$ and $dt_2$ divided into smaller integration time steps.

![Figure 3.5. OPAL integration and simulation time steps](image)

Figure 3.5. OPAL integration and simulation time steps
As previously stated, OPAL abstracts and sets defaults for many ODE simulation parameters. This gives the programmer more intuitive methods to adjust an application and hides some of the lower-level programming details. For example, to control collision responses with OPAL, the programmer simply assigns material characteristics, such as "bounciness" and "hardness", to each object. Using ODE without OPAL, the programmer would have to develop the code for all the logic within the collision callbacks which are invoked whenever a collision event occurs. The contact points would have to be passed to the dynamic simulation where a temporary contact joint would need to be created. The programmer could then create a desired effect by adjusting the ERP and restitution parameters of the joint constraint equations Eqn. (3.1), (3.2), (3.3). This example shows how OPAL greatly simplifies the amount of programming that needs to occur in creating an application. Even though OPAL abstracts and hides many details, most of the adjustable ODE parameters are still accessible by the programmer.

3.3.4 Model Format

The virtual application framework provided by VR Juggler allows graphics to be rendered using OpenGL or a variety of scene graphs. For simplicity, OpenGL was selected as the graphics rendering API. To display the three ODE primitives of a sphere, box, and capsule, the OpenGL Utility Toolkit (GLUT) [54] was used. More complicated CAD model geometry is displayed using the Wavefront OBJ file format.

The OBJ file format was chosen because it is a relatively standard way of storing 3-D model surfaces composed of triangles or higher order polynomials. By using a standard file format, CAD data is easily converted to a form that the application can load. The OBJ file format also stores the same polygonal data as ODE's triangle-mesh shape class. This allows ODE to access the appropriate triangle-mesh data using an OBJ loader to read the initial file as opposed to writing a completely separate loader. ODE requires four pieces of information to create a mesh shape:

1. the number of vertices
2. the number of triangles
3. an array of indexed vertex data
4. an array of indexed triangle data

Many OBJ loaders store data in a redundant manner or index it differently than the ODE triangle-mesh shape class. Several OBJ loaders were investigated in an attempt to find one that stored triangle information in a format that closely resembled ODE’s requirements. The GLM library, included with GLUT version 3.7 [54], contains an OBJ loader with such functionality. The GLM library provides the ability to load, display, and edit OBJ models. This library was selected because it contains a variety of robust model loading options and is simple enough to easily program with. The GLM library also provides data in a form that matches ODE’s mesh data requirements with a small amount of vertex and triangle index manipulation. Figure 3.6 shows three different display modes for an OBJ teapot using the GLM library. Flat shading displays the actual polygon data representing the teapot while smooth shading provides a model representation closer to real life.

![Image of teapot in different display modes](image_url)

**Figure 3.6. GLM library OBJ loading options**

### 3.4.1 Configuration and Model Types

This application uses two file formats for storing model data: XML files and OBJ files. The XML files, provided by OPAL, contain all the physical properties of an object such as initial position/orientation, density, surface friction, surface hardness, etc. The OBJ files contain all the polygonal data for displaying graphics and for creating ODE tri-mesh collision shapes. Using the OBJ and XML file types, two model classes were developed (Table 3.2): primitive and mesh. Since the basic functionality of each class is the same, both classes were derived from a single object base class.
Table 3.2. Summary of part class characteristics

<table>
<thead>
<tr>
<th></th>
<th>Primitive Class</th>
<th>Mesh Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML file</td>
<td>Required</td>
<td>Required</td>
</tr>
<tr>
<td>OBJ file</td>
<td>Not Required</td>
<td>Required</td>
</tr>
<tr>
<td>Physics</td>
<td>ODE Primitives</td>
<td>ODE Mesh</td>
</tr>
<tr>
<td>Graphics</td>
<td>GLUT</td>
<td>GLM Library</td>
</tr>
</tbody>
</table>

The primitive class only requires an XML file to describe the data for a single primitive, multiple primitives, or a group of composite primitives. Primitive objects use ODE primitives for physical representation and GLUT shapes to render graphics. The mesh class requires both an XML and OBJ file to completely describe an object. Mesh objects use ODE tri-meshes for physical representation and the GLM library to render OBJ files for graphics.

Figure 3.7 shows the basic application infrastructure. The first step to initializing the application is loading configuration files. JCCL provides the ability to configure VR Juggler applications by storing information in XML-based text files. These files can be edited with a Java-based GUI called VRJConfig prior to runtime allowing the user to configure an application without editing source code. The two configuration files used for this application are the standard sim.base.jconf and a custom file containing specific model and haptic information.

The standard configuration file contains information relating to the graphic view ports and simulated interaction devices. The custom configuration file provides model information such as the model directory and the number of models. Specifics relating to each model such as the part’s name, scaling factor, and type are identified. Once the part information has been read, the appropriate XML and OBJ files are then loaded based on the particular model type. Also contained in the custom configuration file is the number of PHANTOM haptic devices to be used. Dynamic memory was used to initialize the data structures for each PHANTOM so potentially any number of devices can interact with the application.
Separate threads were implemented for the graphics, physics, and haptics loops to eliminate computational bottlenecks. After loading the haptic and model configuration data, each thread is initialized and then launched. Since keyboard input and graphics output to a computer monitor only need to occur at graphics rates, interaction with these devices is controlled from the graphics loop. Interaction with N number of PHANTOM devices is accomplished in the haptics loop. Since the threads are simultaneously running at different rates, special care is taken to pass data in a thread safe manner.

3.4.2 Virtual Coupling and Object Manipulation

Haptic devices can be divided into two types based on their mechanical behavior: impedance and admittance devices. Impedance devices read displacements and generate forces while admittance devices read forces and generate displacements. The OpenHaptics™
Toolkit selected for this research currently only supports impendence style devices such as the PHANTOM hardware.

The method used for simulating forces in this research is known as “virtual coupling”. Virtual coupling is a widely used technique known to enhance haptic stability when exchanging forces with a dynamics simulation [34] [44]. The method provides a layer of indirection between the haptic device and the simulation. The indirection is necessary since the PHANTOM devices are impendence style haptic devices which mean that forces are not directly available as inputs to the simulation [44]. Instead, a layer of indirection must be introduced to calculate forces based on a positional change from the device. These forces are then sent to both the physics simulation and the device. Conceptually, the layer of indirection is applied by coupling the haptic device’s end-effector (end of the device kinematic chain) to the manipulated dynamic object using a spring damper system shown in Figure 3.8. The end-effector’s representation in the virtual environment is referred to as the virtual haptic handle.

![Figure 3.8. Virtual coupling spring damper system](image)

3.4.2.1 **Virtual Spring Equations**

The virtual spring system consists of both a linear and angular spring with corresponding dampers. This setup allows both force and torque information to be transferred between the dynamic object and the haptic device. The linear force is
proportional to the offset displacement $d$ between the virtual haptic handle and the spring’s attach point to the dynamic object. The torque is proportional to the offset orientation angle $\theta$ between the haptic handle and dynamic object. The force and torque values are calculated by equations Equ. (3.4) and (3.5) shown below:

\[
F_{\text{spring}} = k_L d - \delta_L v \quad \text{Equ. (3.4)}
\]
\[
\tau_{\text{spring}} = k_R \theta - \delta_R \omega \quad \text{Equ. (3.5)}
\]

where

$k_L, \delta_L =$ linear spring’s stiffness and damping constants
$k_R, \delta_R =$ rotational spring’s stiffness and damping constants
$d, \theta =$ spring offsets for linear distance and rotational angle
$v, \omega =$ dynamic object’s relative linear and angular velocity

The damping constants $\delta_L$ and $\delta_R$ are required so the spring system can reach equilibrium. Equilibrium occurs when the haptic handle is not moving and forces affecting the dynamic object are balanced. Without dampers, the applied spring forces would cause the manipulated object to constantly overshoot the equilibrium position and orientation. This causes a vibration effect as the spring constantly pulls the dynamic object back and forth across equilibrium but never actually reaches it.

The haptic end-effector controls the position and orientation of the virtual haptic handle shown in Figure 3.8. Movement made by the device influences the amount of spring offset and generates a force/torque on the dynamic object. In turn, an opposite force is generated for the haptic handle which is sent to the device to simulate haptic force feedback. The spring force supplied by the haptic handle is just one force applied to the dynamic object. Other forces from the physics simulation are also accumulated such as collision impulses, friction, and gravity. These forces indirectly affect those sent to the haptic device allowing the user to feel the dynamic part interact with the simulation environment.

A nice feature of the virtual coupling method is that the spring constants can be defined differently for the forces applied to the haptic device and the virtual object. This
feature allows separate tweaking of the physics simulation and the haptic device’s force feedback. The virtual spring stiffness is set as high as possible while still maintaining a stable simulation with realistic behavior. The haptic device spring stiffness is adjusted to scale the strength of the force feedback. Higher haptic spring stiffness corresponds to sharper force feedback when objects collide but more viscosity when freely manipulating objects. The virtual coupling method requires some trial and error to obtain desired results as there is currently no scientific method to produce the most realistic simulation.

### 3.4.2.2 OPAL Tools for Virtual Coupling

As previously noted, OPAL abstracts many of the lower-level details by providing intuitive data structures. Two such structures, a volume sensor and a spring motor, were used to apply the virtual coupling method. A volume sensor is an OPAL data structure that queries the simulation environment using a “volume” solid to determine which objects have collided with the sensor. A volume solid is a special dynamic object that does not interact with the environment, but rather just returns a list of intersected objects. Although any ODE shape can be used as a volume sensor, this research only utilized a sphere shaped sensor.

An OPAL spring motor is a data structure that applies forces and or/torques to simulate a linear and/or torsional spring. The spring is attached to a rigid body in order to bring the object to a desired position and/or orientation. The programmer then supplies the spring motor with the spring stiffness and damping constants of equations Equ. (3.4) and (3.5) along with a desired position and/or orientation.

Together the volume sensor and spring motor provide all the necessary tools for selecting and manipulating virtual objects. The volume sensor is attached to the virtual haptic handle so its motion corresponds to the haptic device. This allows the haptic device to be used as a 3-D cursor for selecting objects. After pressing button one of the PHANTOM device, the volume sensor queries the environment for solids that are colliding with the sensor’s shape. If the sensor indicates that an intersection has taken place, one end of the spring motor is attached to the collided object. In the case where the volume sensor returns more than one intersected object, the spring is attached to the first object. The other end of the spring motor is attached to the virtual haptic handle to complete the coupling.
3.4.2.3 Realistic Part Manipulation

Since real life objects behave differently depending on where they are grasped, a realistic simulation needs to account for arbitrary grasp locations. The last location of the virtual haptic handle just before an object is selected is referred to as the object’s selection point. The attachment of the virtual spring to the dynamic object at the selection point is critical for realistic part interaction. In the absence of a selection point, object manipulation is performed about the object’s COG. If the haptic simulation does not account for arbitrary locations of the selection point, the simulation will lack realism as shown by the scenarios in Figure 3.9.

![Diagram showing unrealistic and realistic physics for a manipulated part]

**Figure 3.9. Unrealistic and realistic physics for a manipulated part**

The problem of implementing offset object manipulation is best demonstrated with a long slender object where the selection point is a noticeable distance away from the dynamic object’s COG. Scenario 1 of Figure 3.9 shows the case where object manipulation occurs about the COG regardless of the selection point location. When a collision event occurs with the static object, rotation is around the center of gravity as opposed to the point where the
object was grabbed. This can create both a visual and haptic discrepancy because the part rotation is not representative of real motion. Scenario 2 shows a more realistic case object rotation is about the selection point. When the collision event occurs in this case, the part motion simulation is more realistic. By default, when a spring is connected to an OPAL solid, its rotation is about the COG. However, OPAL allows for an attach offset that stores the position of the spring attach point relative to the solid’s COG.

Another common problem occurs when the dynamic object snaps its COG to the virtual haptic handle. The ideal circumstance is for the dynamic object to only demonstrate motion when the haptic handle moves. To avoid this snapping effect, the first desired position is set as the current position of the dynamic object when the object is selected. Following desired positions are set by the virtual haptic handle taking into account the original orientation and position offsets between the solid and the handle. This method ensures natural motion when selecting objects, manipulating objects freely, and colliding objects with the virtual environment.

3.4.3 Mapping Haptic Device

There are many technical issues to address when mapping the haptic device workspace to the simulation environment. The OpenHaptics™ toolkit provides device mapping utilities; however, complications occur in implementing these methods since VR Juggler requires multiple view frustums for stereo imaging. Figure 3.10 provides the transformation pipeline, used in this research, for mapping the original device coordinates \((x_0, y_0, z_0)\) to the simulated camera view coordinates \((x_c, y_c, z_c)\).

![Transformation pipeline for mapping haptic device](image)

**Figure 3.10. Transformation pipeline for mapping haptic device**
The first transformation matrix is applied to the original device coordinates \((x_0, y_0, z_0)\) to take into account the unit differences between the millimeters units of device and default feet base units of the VR Juggler environment coordinates \((x_{VE}, y_{VE}, z_{VE})\). The matrix is applied so that the virtual haptic workspace approximately coincides with the graphics view frustum. Different scale and translation factors are used for separate devices to create the same virtual workspace.

The virtual coordinates \((x_{VE}, y_{VE}, z_{VE})\) are then multiplied by the camera position matrix to yield the virtual workspace in camera view coordinates \((x_c, y_c, z_c)\). Without this second transformation, navigation through the virtual environment leaves the virtual workspace in its initially placed position/orientation. The transformation ensures the virtual haptic workspace always stays within the user’s view. VR Juggler’s Gadgeteer device manager was used to proxy the simulation camera’s transformation matrix.

### 3.4.4 Application Flowchart

Figure 3.11, of the following page, presents the application flowchart. As previously noted, the application is launched into three separate threads once the configuration and model files have been loaded. The threads are implemented by priority, starting with the high priority haptic thread, then the physics thread, and finally the graphics thread.

#### 3.4.4.1 Haptics Thread

The haptics thread is responsible for communicating with the haptic device/s. It reads position/orientation and button activation data from the haptic device/s and sends a calculated force to the device/s. As seen in Figure 3.11, OpenHaptics™ launches a separate high priority, high frequency (~1000 Hz) servo loop thread for interacting with the PHANTOM device/s. Each device is assigned its own set of haptic frames where most haptic operations are performed. Within each frame, state information is guaranteed to be consistent for the particular device. Interaction with multiple devices requires only a single servo loop thread. With every pass of the servo loop, the application steps through each device executing its haptic frame.
Figure 3.11. Application Flowchart
OpenHaptics™ provides a scheduler to manage interaction with the servo loop thread. The scheduler allows access to the servo loop using both synchronous and asynchronous callbacks. Synchronous callbacks are best for taking thread safe snapshots of haptic device state information. Figure 3.11 shows a synchronous callback invoked in the physics thread.

Asynchronous callbacks are implemented each pass of the servo loop and are used to control the events that occur inside the haptic frames. The asynchronous callback for this application loops through the PHANTOM devices by toggling which device is current. If an object has been selected, a calculated force is sent from the physics thread to the servo loop for rendering to the current haptic device. A minimal number of tasks are performed by the asynchronous callback in an attempt to maintain the high-speed haptic update rates.

3.4.4.2 Physics Thread

The physics loop uses OPAL/ODE to perform all the collision detection and dynamics calculations for simulating realistic part behavior. These calculations provide the graphics loop with part position/orientation information used to display visual feedback and the haptic loop with a force vector used to render force feedback. The speed and accuracy of the physics thread is controlled by the integration time step size. Frame rate values ranging from 100 to 1000 Hz were tested. By default, OPAL does not perform physics calculations in a separate thread from the main application. In an attempt to free the physics calculations from graphic delays, a separate thread was created using the VPR library to manage the physics simulation. VPR provides a cross-platform, object oriented abstraction layer for many common operating system features such as threading.

Figure 3.11 shows the basic operation of the physics thread. The physics loop starts each pass by getting the time elapsed since the previous frame. Opal then simulates the physics, an amount of time ahead equal to this increment, for the following frame. The physics thread then loops through each haptic device querying the simulation environment for GRAB and RELEASE events. To query the virtual environment, the application uses an OPAL volume sensor based on the haptic device’s position information obtained from a thread safe synchronous callback.
Once an object is selected, the virtual coupling method of section 3.4.2 is implemented and a force vector calculated for the haptic device. OPAL provides the ability to access physics simulation data following each integration time step using a post event handler callback. However, problems occur if the physics data is being written while the other threads are trying to access it. The VPR library solves this problem by providing a Mutex to lock and unlock access to data shared between threads. This application uses Mutex data synchronization by locking access to physics information while ODE is integrating and releasing access after ODE is done. This allows the other threads to access the physics information unless ODE is currently writing the data. Once an object is released, the virtual spring coupler is disabled and forces are no longer sent to the haptic device.

3.4.4.3 Graphics Thread

VR Juggler is responsible for launching the graphics thread which operates as fast as the PC system can render the entire graphics scene. Only actions that need to occur at graphics rates are executed in the graphics thread. Each time through the graphics loop, collision checking is performed in VR Juggler’s preframe to detect intersection of the virtual haptic device with an object. Querying of the environment is done using an OPAL volume sensor based on the haptic device’s position information from the synchronous callback invoked in the physics thread. Information is kept thread safe by applying a Mutex to lock access to the haptic state information while either the graphics or physics threads are accessing it. If the volume sensor intersects an object, the object’s color is changed to notify the user of an intersection. After performing an intersection query for each PHANTOM device, the virtual scene is then graphically rendered.
CHAPTER 4. SYSTEM RESULTS AND ANALYSIS

An evaluation of the application’s performance as a virtual assembly tool is presented in this chapter. Before analyzing these scenarios, a description of the program functionality and interaction is accomplished. The application’s collision detection capabilities are then evaluated for various bolt and hole clearance situations. Scenarios including interaction with many dynamic objects and the ability to interact with large complex mesh models are also analyzed.

4.1 Application Interaction

Before the application can load and interact with CAD geometry, the model data must first be formatted to the XML and OBJ file formats introduced in chapter 3. This section covers how the data is created for each file type as well as the application’s basic interaction. Different interaction modes are explored to determine which are most intuitive for a realistic assembly application.

4.1.1 Model Preparation

The primitive model class has an associated XML file containing collision shape and physical property information. Primitive collision shapes were positioned using 3DSMax and exported to an XML file using a plug-in script supplied by OPAL. This method allows the visual placement of primitive shapes as opposed to blindly editing an XML file. The user can then set an object’s physical properties by editing the XML file or allow OPAL to assign defaulted values.

The mesh model class must have an associated OBJ file to define a collision shape and render a graphical representation. Figure 4.1 illustrates the two OBJ conversion paths utilized by this research. The first path converts an existing JT file to an intermediate VRML file format using Vis Mockup 5.1. From here, the Nugraf® Rendering System converts the model into the final OBJ file format. A similar path was identified for SolidWorks 2004 so models created with the CAD package could be utilized in the application. Pro/Engineer® CAD modeling software has the ability to directly export to an OBJ file so no file conversions were required. Both Nugraf® and Vis Mockup offer many model translation
options, some of which were utilized in this research including the ability to reduce a model’s polygons through decimation. Although the previous conversions were used, any conversion path is allowed so long as the correct final format is created.

![Diagram of OBJ model conversion paths]

**Figure 4.1. OBJ model conversion paths**

Figure 4.2 shows some OBJ CAD models and their corresponding physical primitive representations created with 3DSMax. The fidelity of such parts’ physical behavior is limited to the accuracy in which the primitives represent the original CAD model. To create a precise representation can take a large amount of model preprocessing work.

![OBJ model conversion paths]

**Figure 4.2. Primitive representation (left) and OBJ tri-mesh representation (right)**

### 4.1.2 Manipulating and Selecting Models

Once the application has started, the user is able to haptically interact with the parts loaded into the virtual environment. Figure 4.3 shows the application and a subject utilizing stereo glasses to display 3-D images and two PHANTOM Omnis to haptically simulate dual-handed assembly tasks. Although stereo vision and dual PHANTOM devices are illustrated,
the application is easily configured to operate without stereo and with a single PHANTOM device.

![Image of PHANTOM device interaction](image)

**Figure 4.3. The application using stereo vision and dual PHANTOM interaction**

To select the virtual parts, the user grasps the haptic stylist or handle which controls the location of the virtual haptic handle, represented by a sphere. A part can have three different states, each of which is represented by a different color (Figure 4.4):

1. unselected: neutral gray
2. intersected by virtual haptic handle: bright green
3. selected: dark green

By default, parts are drawn a neutral gray if they are not selected (Figure 4.4 (a)). Virtual objects change to a bright green color (Figure 4.4 (b)) to provide the user with a visual cue that the haptic handle has intersected them. An object can be selected while intersecting the haptic handle by pressing button one of the PHANTOM device. To indicate the selection, the object undergoes yet another color change to a dark green (Figure 4.4 (c)). Once an object is selected, the haptic handle and virtual object are connected together with the virtual spring coupling method discussed in section 3.4.2. Forces are then sent back and forth between the PHANTOM device and the manipulated object to simulate the part’s interaction with the virtual environment.
More than one device can be used to simultaneously manipulate a single part (Figure 4.4 (d)) or multiple parts (Figure 4.5). In both cases, a separate virtual spring coupling is created between the virtual object and each device. Force feedback is separately sent to each PHANTOM in order to simulate forces generated by the adjacent device and the manipulated part’s interaction with the simulation environment.

**Figure 4.4. A virtual wrench (a) before colliding with haptic handle, (b) while colliding with haptic handle, (c) after being selected, (d) and during dual Omni manipulation**

**Figure 4.5. Dual haptic interaction with multiple parts**

### 4.1.3 Interaction Modes

Several interaction modes can be created by applying different ODE physical properties. For example, objects in the virtual environment can be set to either static or dynamic. Static bodies are objects which have a collision shape but no associated dynamics...
body. Dynamic objects can still interact and collide with static objects, but a static object itself always remains immobile. The two interaction modes explored in this research are:

- Mode 1 (dynamic mode) - All bodies are considered dynamic throughout length of the simulation.
- Mode 2 (static mode) – All bodies are set to static until selected by a haptic device, after which, an object is then changed to dynamic. Upon releasing the object it is returned to static.

Figure 4.6 provides an example illustration of first mode in which a tower is created from a set of dynamic blocks. This simulation provides an evaluation on the number of dynamic objects that can stably interact with each other while still maintaining the required graphics, physics, and haptics update rates. Running on a DELL Precision 670 dual processor Xeon™, various simulation parameters were tweaked to allow 52 dynamic blocks to statically rest on top of each other. A useful OPAL parameter that was utilized is the sleepiness of a dynamic object which disables a body if its velocity stays within a threshold for a certain amount of time. This parameter saves computation time since the dynamics solver does not perform calculations for the disabled bodies.

![Stable tower of dynamic blocks](image)

**Figure 4.6. Stable tower of dynamic blocks**

Results of the simulation showed stable haptic interaction while each block exhibited real physical properties for the effects of gravity, surface friction, and collisions with other blocks. Simulating towers with more that 52 blocks created problems with the haptic feedback as the physics engine became overwhelmed with calculations. While the physics
engine may slow down enough to compute the correct physics, the decreased rate in forces sent to the device demonstrated faulty and error prone haptic feedback.

While the first mode provides the most realistic physics simulation, it does not necessarily provide the most intuitive interaction for virtual part assembly. For example, as parts collide with the manipulated object, they can reach undesired positions and/or orientations leading to situations that complicate assembly. Parts must also rest against static surfaces in order to reach equilibrium so assembly operations can be performed. The second mode alleviates these problems and provides more intuitive interaction by setting non-manipulated parts to static. This mode allows a part to be positioned in a desired manner and then held static while other parts are assembled to it. Gravity still affects the manipulated objects in the form of weight but does not influence static objects.

The second mode also allows more complex assemblies to be analyzed since the amount of physics calculations are much less for static objects than dynamic ones. This is because static objects do not require dynamics calculations and collisions are minimized since static objects can not collide with each another. ODE’s collision engine is also optimized for detecting dynamic to static collisions [55]. Since the second method provides more intuitive interaction and is computational less expensive than the first, method 2 was utilized for the remainder of application testing.

4.2 Program Testing

Different assembly scenarios were developed, using both primitive and mesh parts, to evaluate the application’s collision detection and physical response accuracies. Analysis of ODE’s collision detection and the application’s ability to interact with large mesh models is also achieved.

4.2.1 General Assembly

Results of primitive-to-primitive assembly showed accurate collision detection with very realistic physical responses. Primitive-to-mesh assembly interactions also demonstrated accurate collision detection with moderately accurate physical responses. However, some variation from real life physics is expected since OPAL only approximates certain mesh shape physical properties. For example, an object’s COG location is set to the center of the
mesh shape’s bounding box. Physical properties are corrected by editing the object’s XML file but at the expense of some model preprocessing.

Mesh-to-mesh assembly presented accurate collision detection while most of the time realistic physical responses were not demonstrated. Problems with physical responses occurred most frequently when mesh objects had continuous contact with each other, while situations with momentary contacts were occasionally simulated correctly. Error prone behavior included excessive surface stickiness between colliding meshes and occurrences where mesh shapes sank into one another. Once two mesh objects have extensively penetrated one another, the physics loop becomes overwhelmed and can no longer supply force information fast enough for haptic rates.

Primitive-to-primitive assembly scenarios were used to test the value provided by haptic force feedback since these interactions provided the most realistic physical responses. The scenario of assembling a primitive square bolt into a primitive square hole (Figure 4.7) provides a good evaluation of haptics by using the PHANTOM Omni with and without haptic force feedback. A relatively tight clearance of a 0.99 thick bolt and a 1.0 thick hole was used to create a moderately difficult assembly task.

![Figure 4.7. Assembling primitives to evaluate force feedback](image)

In general, assembly with force feedback offered more natural interaction by providing instantaneous cues of part collisions. Without force feedback, the user was limited to delayed visual cues that a part had stopped following the device’s motion. Force feedback also helped guide user motion once a bolt had been partly inserted into the hole. Without force feedback the user’s hand was free to travel where the simulated part could not go.
After experimenting with the application, it is quite evident that the lack of torque feedback when visually simulating rotational collision responses creates problems. The previous bolt/hole assembly scenario provides a good example of this. Reaching the required orientation for placing the bolt into the hole is difficult since torque force feedback is not provided to help guide the user’s hand. The user is limited to visual cues to explore different bolt/hole orientations. Attempting to change the bolt’s orientation after it has been placed in the hole also does not provide the user with realistic force feedback. This behavior can create the illusion that certain device motions do not affect the simulation.

4.2.2 Collision Detection Testing

One important aspect when performing assembly and maintenance operations is the amount of clearance between parts and/or tools. An ideal simulator should not allow parts to be assembled if the amount of clearance is below a certain threshold as experienced in real life. A “drop test”, shown in Figure 4.8, was developed to evaluate ODE’s collision detection capabilities.

![Figure 4.8. Bolt drop collision test](image-url)
The bolt drop test provides an evaluation of assembly clearances by testing ODE’s ability to detect collisions between various sized bolts and a block with a hole. The block was kept static while the dynamic bolt dropped into the hole. Since this application requires a specific position and orientation of parts to assemble them, the ease in assembling parts can vary from user to user. To eliminate human error and create a repeatable test, the only force acting on the bolt was gravity. The worst case collision detection scenario was tested by perfectly aligning the bolts position and orientation required for assembly with respect to the hole. ODE collision callbacks were used to notify if and when a collision registered.

Separate tests were performed to evaluate collision detection for primitive-to-primitive, primitive-to-mesh, and mesh-to-mesh scenarios. The amount of clearance was calculated by the difference in the hole thickness $T$ and the bolt thickness $t$ as seen in Equ. (4.1) below.

$$\text{Clearance} = T - t \quad \text{Equ. (4.1)}$$

Smaller and smaller clearances were tested until a collision event was detected. To accomplish this task, a hole with a one unit thickness was held constant while the bolt’s thickness increased until a collision registered. Figure 4.8 gives a summary of the tested geometry while Table 4.1 presents the collision detection results for both float and double precision numbers. Double precision could only be tested for the primitive-to-primitive collision scenario since the GLM library can only compile with floats.

<table>
<thead>
<tr>
<th>Colliding Geometries</th>
<th>Clearance at Which Collision was Detected</th>
<th>Precision (Float or Double)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive Square Bolt Primitive Square Hole</td>
<td>$8.0 \cdot 10^{-8}$</td>
<td>Float</td>
</tr>
<tr>
<td>Primitive Square Bolt Mesh Square Hole</td>
<td>$1.0 \cdot 10^{-7}$</td>
<td>Float</td>
</tr>
<tr>
<td>Primitive Circular Bolt Mesh Circular Hole</td>
<td>$2.0 \cdot 10^{-2}$</td>
<td>Float</td>
</tr>
<tr>
<td>Mesh Square Bolt Mesh Square Hole</td>
<td>$9.0 \cdot 10^{-7}$</td>
<td>Float</td>
</tr>
<tr>
<td>Primitive Square Bolt Primitive Square Hole</td>
<td>$2.0 \cdot 10^{-16}$</td>
<td>Double</td>
</tr>
</tbody>
</table>
Analysis of Table 4.1 shows ODE’s collision detection is rather accurate. The square hole/bolt clearance results vary a small amount but overall demonstrate collision detection accuracy to almost exact float or double precision. C++ float numbers contain 7 digits of accuracy while doubles contain 15 digits. The lower accuracy shown by the circular hole/bolt is due to some precision loss when converting the original CAD data to the OBJ tri-mesh data format. Most of today’s CAD software packages use an industry standard method known as NURBS to store curve and surface data. The OBJ circular hole is only a polygonal approximation of the original NURBS curved surface representation of the hole.

The drop test results demonstrate that ODE can detect collisions with tighter part clearances than parts are typically assembled with in real life. However, there are many factors besides collision detection that determine whether simulated parts actually fit together. Parameters that influence a part’s physical response, such as the integration time step, instruct the simulation what to do after a collision event has occurred. For example, if the integration step size is too large, two parts may penetrate each other past a correctable amount. Parts can also be forced together if the user applies a force greater than the contact joint penalty force used to correct penetrations. This is especially true for close fitting parts where only a small penetration depth is detected.

4.2.3 Model Loading Tests

After collision detection analysis, large complex mesh models were loaded to test the application’s interaction capabilities and identify limiting areas of the system. Collisions were minimized by applying the static interaction mode and limiting the simulated parts, besides the single complex mesh, to relatively simple models comprised of primitives. Single PHANTOM testing occurred on both the Dell™ Inspiron 8600 laptop and the Dell™ Precision 670 PC while stereo vision was only implemented with the Precision PC.

Frame rates were taken from each of the graphics, physics, and haptics loops to determine which thread was the limiting factor. The graphics and haptics threads were determined to fail if either fell below the rates required by the human body’s senses of sight (30 Hz) and touch (1000 Hz) respectively. The physics loop was determined to fail if its
frame rate went below the integration time step size, which was set at 0.01 seconds (100 Hz) for this testing.

Testing results without stereo showed the Precision PC could handle models up to 300,000 triangles while the Inspiron laptop could handle models up to 60,000 triangles. For models larger than these, the graphics thread could not render the scene at the required rate of 30 Hz. Implementing stereo vision cut the model size that could be graphically rendered by the Precision PC directly in half to around 150,000 triangles. Dual haptic assembly dropped the maximum number of rendered polygons a slightly marginal amount. Testing of larger models, up to 1,000,000 polygons, still maintained the required physics and haptic frame rates even though the graphic rates could not be completed fast enough.

Although the graphics thread was identified as the limiting factor for the previous test, other scenarios provided circumstances where the physics loop fell below acceptable rates. Since the graphics and haptics threads both depend on calculations performed in the physics loop, a drop in the physics rates influences the entire application. The ability of the physics thread to maintain desired integration speeds was identified to closely relate to the number of contact points generated for colliding objects. Scenes with many dynamic objects are not necessarily required to overwhelm the physics thread with contact calculations. For instance, a single complicated mesh can easily create enough contact points with primitives or another mesh shape to slow the integration speed. Since OPAL does not allow the number of contacts to be directly queried, analysis on the exact number of contacts required to slow system performance is not achieved. The haptics thread was not identified as the limiting factor for any testing. This is probably because the haptics loop is given the highest priority thread and is responsible for a minimal amount of tasks.
CHAPTER 5. CONCLUSIONS AND FUTURE WORK

This research accomplished the stated goal of investigating the feasibility of an affordable haptic desktop system for evaluating assembly operations. The application behind this investigation combined several software packages including VR Juggler, OPAL/ODE, OpenHaptics™, and OpenGL/GLUT/GLM to explore the benefits and limitations of combining physically-based modeling with haptic force feedback. Affordable hardware was identified to provide haptic force feedback and stereo vision implemented to provide a more immersive experience. In testing the application, the system’s collision detection and the complexity of loadable models were analyzed.

5.1 Conclusions

After using the application and evaluating different scenarios, some conclusions can be drawn about the system as a means for evaluating assembly operations and the tools used.

1. In general, haptic feedback even without torque does help in assembling virtual parts. Even simple tasks, such as placing a bolt into a hole, feel more intuitive when force feedback is supplied to the user’s hand. Although not tested, attempting the same assembly tasks with a standard mouse and keyboard would take multiple key presses and commands.

2. ODE appears to detect collisions quite well as testing showed primitive-to-primitive, primitive-to-mesh, and mesh-to-mesh collisions were detected with almost exact float or double precision. However, problems sometimes did occur in simulating the correct physical response after detecting the collisions.

3. ODE’s physically-based modeling was found to have limitations in that it can not accurately simulate physical responses between interacting mesh geometries. While primitive-to-primitive and primitive-to-mesh physical responses were fairly realistic, accurately representing CAD geometry with primitives requires a large amount of preprocessing. Until mesh-to-mesh physical responses have been perfected, the problem of assembling arbitrary CAD geometry using physically-based modeling will not be solved.
4. Although the application can not accurately simulate physical responses between colliding mesh geometries, the system can still be used to evaluate the feasibility of some assembly and maintenance operations. A single mesh shape can be used to represent a rather complex part or subassembly while primitives used to approximate smaller parts and/or tooling. For example, Figure 4.5 illustrates a primitive representation of a wrench that can be used to evaluate awkward reach angles around the complicated mesh hitch assembly.

5. Different assembly circumstances were created to analyze limiting areas of the application. As expected, the graphics loop limitation occurred when the number of polygons reached a level where the system can no longer render the scene at a rate of 30 Hz. The physics thread’s ability to maintain the requested integration speed was found to directly relate to the number of contacts generated between colliding geometry. Testing did not identify any scenario where the haptics thread was a limiting factor.

6. An interaction mode where parts are kept static until selected was identified as the most natural and computationally efficient. Dynamically simulating every part in a virtual scene creates the most realistic physics simulation but complicates part assembly operations and slows system performance by increasing the required physics calculations.

7. Although OPAL abstracts and sets defaults for many ODE simulation parameters, many variables must still be tweaked to achieve desired simulation behavior. Certain changes require some trial and error as there is no scientific method to achieve the best result. Obtaining a desired simulation with the current status of physic engines appears to be somewhat of an art as opposed to an exact science.

5.2 Future Work

Although this research succeeded in accomplishing the goal of creating an affordable haptic desktop system for evaluating assembly operations, there are several areas that could benefit from improvements. Improvements to the hardware would provide the application with a greater sense of immersion and higher fidelity force feedback, while changes to the
software could allow the application to handle more complex CAD models. Some suggestions for future improvement are included below.

1. A more realistic simulation could be achieved by using higher fidelity haptic devices. This application utilized 6-DOF freedom input devices that only provide 3-DOF or positional force feedback. A higher fidelity simulation could make use of a 6-DOF force feedback device produced by SensAble. Since OpenHaptics™ provides the ability to send forces to a 6-DOF device, only a small amount of programming would have to occur to integrate such a device. Providing force feedback in gripping objects as well as force feedback to the user’s hand when part collisions occur could also provide a higher fidelity simulation. This option is currently available by combining Immersion’s CyberForce® and CyberGrasp™ hardware. However, it is noted that these devices are currently too expensive for wide use in industry.

2. The active stereo method applied in this research could also benefit from head tracking. Without head tracking, stereo images are drawn with respect to a certain view and as the user strays from this perspective images can appear skewed or distorted. Implementing head tracking would convey more accurate images by drawing to the user’s exact perspective at all times. Future research would have to identify an affordable tracking system for use with a desktop system.

3. The application presented in this research also estimates or sets default values for simulated objects’ physical properties. While the user can manually correct these properties by editing the model’s XML file, this can lead to large amount of preprocessing. Depending on the CAD software package, some of this information may be present in the original model file. For use as a more effective design tool, the application could extract the physical properties from the original CAD model data and apply them to the simulated parts.

4. Combining constraint-based modeling with physically-based modeling may also offer a more realistic simulation. As application testing showed, physically-based modeling is currently limited by the complexity of situations it can handle (i.e. no mesh-to-mesh physical responses). Cases where physically-based modeling is not good enough could be simulated, at least partially, with constraint-based modeling.
Constraint data could also be extracted from the original CAD assembly files to minimize preprocessing.

5. Although no current physics engine can perfectly handle collision responses for arbitrary meshes, physics engines that predict collisions may be better suited to simulate such behavior. In order to simulate collisions, ODE applies penalty forces to objects that have penetrated each other at the end of each time step. Other physics engines use methods to predict when a collision will occur, even between time steps, preventing objects from ever penetrating. These methods may be more appropriate for handling mesh-to-mesh collisions since ODE seems to experience problems when meshes penetrate each other.
BIBLIOGRAPHY


